Enterprise Integration of Smart Objects using Semantic Service Descriptions

Matthias Thoma*† Torsten Braun † Carsten Magerkurth*
*SAP (Switzerland) Inc., Althardstrasse 80, 8105 Regensdorf, Switzerland
matthias.thoma@sap.com, braun@iam.unibe.ch, carsten.magerkurth@sap.com
†Communication and Distributed Systems, University of Bern, Neubrückstrasse 10, 3012 Bern, Switzerland

***Abstract*—Integrating physical objects (smart objects) and enterprise IT systems is still a labor intensive, mainly manual task done by domain experts. On one hand, enterprise IT backend systems are based on service oriented architectures (SOA) and driven by business rule engines or business process execution engines. Smart objects on the other hand are often programmed at very low levels. In this paper we describe an approach that makes the integration of smart objects with such backends systems easier. We introduce semantic endpoint descriptions based on Linked USDL. Furthermore, we show how different communication patterns can be integrated into these endpoint descriptions. The strength of our endpoint descriptions is that they can be used to automatically create REST or SOAP endpoints for enterprise systems, even if which they are not able to talk to the smart objects directly. We evaluate our proposed solution with CoAP, UDP and 6LoWPAN, as we anticipate the industry converge towards these standards. Nonetheless, our approach also allows easy integration with backend systems, even if no standardized protocol is used.**

I. Introduction

In the intersection between enterprise IT systems and the embedded world, also known as the Internet of Things (IoT) or Cyber-Physical Systems (CPS) one can currently observe two major trends: (i) Standardization of network protocols, in particular IP-based protocols (6LoWPAN) [1] and (ii) use of high level application protocols, in particular CoAP [2]. Both developments are driven by the needs of software producers that need standards and higher abstraction levels, thus reducing the gap between how software is traditionally written and the very specialized knowledge currently needed to write applications for wireless sensor networks [3]. They are crucial for an adoption of sensor network technology in industrial applications, where interoperability plays a decisive role.

Going a step beyond interoperability at a very technical level, *semantic integration* of sensor and actuator devices is foreseen as one of the next game changers. In this paper we introduce a novel and industrially applicable semantic service integration framework. Compared to previous work it is not only for describing sensor landscapes as, for example, W3C SSN[1], but is completely usable to describe service endpoints at a low level. Other than WSDL, and existing solutions around WSDL, our solution is fully integrated into the semantic web, and does not use anntotations as SA-WSDL does. Additonally, we provide an quantitative evaluation of our semantic service descriptions on actual hardware, while most of the previous work on semantic integration stays at a high level and has never been running on real hardware.

We first briefly introduce our idea of a semantic service description based IoT-integration platform and then go on by presenting two concepts that allow an easy integration of embedded devices into enterprise environments: semantic endpoint descriptions and Smart Endpoint Generation (SEG). Semantic endpoint descriptions are used to describe the properties of an endpoint and the communication paradigm used, while smart endpoint generation provides an enterprise application with various means to access services provided by smart objects based on the original semantic service description.

[1]http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/

II. Related Work

Enterprise Integration of Wireless Sensor Networks has traditionally been done either by using WSDL [4], HTTP as in the web of things movement [5] or by introducing proxy-like components for a conversion between sensor network protocols and the enterprise side. The idea of describing services in an ontological way is part of the Linked Services movement [6]. Linked USDL [7], which is used in this paper, is one instantiation of Linked Services. Nonetheless, Linked Service approaches are focusing on describing the services at a high level and were not designed to enable low-level interoperability, in contrast to WSDL, which is sometimes supported. RDF as a data format within WSNs has gained some attention recently, especially in the IoT-context[8]. While all these works concentrate on specific aspects, our work can be seen as an umbrella providing an unified way to describe and access sensor services.

Linked Services are not the only way to introduce semantics. Apart from our top-down approach from semantics to endpoints there are also alternative, more endpoint centric, ways: SA-WSDL [9] introduces semantic annotations into the endpoint description. Furthermore microformat based solutions such as hrests [10] introduce semantics into (web-based) service documentation. Closest to our approach is OWL-S [11], which also uses semantic web technologies to model services. OWL-S is tailored towards web-services and does not take business aspects into consideration.

III. Intergation Platform

Our semantic endpoint descriptions and the smart endpoint generation are to be used within a SOA based enterprise integration platform. The main layers of our architecture are shown in Figure 1.

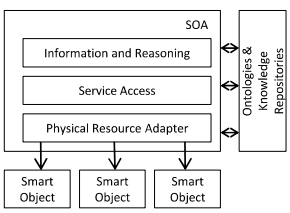

Fig. 1. Layered Architecture

The communication between the smart objects and the integration platform is done via specialized physical resource adapters. They are feeding their knowledge about the respective resources into the knowledge repositories. Access to

services and provisioning of services to enterprise systems is done by the service layer. The smart objects themselves are self-descriptive, thus carrying at least a minimal description of their capabilities with them. The service layer might need to complete the service description by following the URIs or by decompressing (parts of) the description. Furthermore, with the business information and reasoning layer the service descriptions might provide additional higher level services based on SLAs, reasoning (for example ontology based virtual sensors) or the connection of entities and (virtual) sensor data. As an example, one is usually not interested in the temperature of device no. 1321, but in the temperature of some given good, which could be monitored by several sensors. This abstraction, away from the sensing devices, towards the "things" they monitor is one of the key ideas of IoT.

The deployment view of our semantic enterprise integration platform is shown in Figure 3. The Integration Platform Instances (IPI) are deployed to different (geographical or logical) zones and control one ore more Smart Objects (SO).

IV. Semantic Endpoint description

In this section we first argue for the necessity of service descriptions. We then explain Linked USDL and add different endpoints and message representations.

A. Introduction

In traditional approaches for describing services for smart objects like SOAP or TinyWS [12], interoperability comes for a price in terms of computing power, energy consumption and delay. We are following the approach of service descriptions that can be well separated from the actual service and thus allowing XML based SOAP or REST like interfaces, as well as propriatary binary protocols. First, we define the term service description:

> A service description is a description of all essential properties of a given service, as well as the means to access it. A service description is independent of an actual implemented callable service.

It is important to distinguish between service descriptions and the actual implemented (callable) service on a device. The *technical interface* to this callable service is called *endpoint*.

In a semantic IoT integration platform, there are many domain specific properties to be described: For example, among others, Quality of Information parameters, Quality of Service, location, or SLAs. At a service level this information can be used to calculate the resulting QoI of composed services. The ontological link to these values is either attached to the service as a whole or, where applicable, to the endpoint.

In this work we look at three different ways of integration: CoAP-based REST style, SOAP based web services and a custom structured byte based protocol specified in ASN.1 [13]. This allows us to not only specify standard compliant web services, but also allows interoperability at a very low level to demonstrate the integration of custom protocols. ASN.1 is used to quantify the overhead of CoAP and SOAP in comparization to specialized protocols.

B. Linked USDL

We propose to use Linked USDL (LUSDL), as shown in Figure 2, for describing services. LUSDL has the advantage of going beyond the technical interface. It covers not only functional, but also operational and business aspects.

usdl:Service is the main entry point when modeling services. It describes a service such that it can serve as an interface between the service provider and the service consumer. The service description contains functional as well as non functional properties of the service. The functional (technical) properties are described by the interaction protocol (interaction points). The non-functional properties are described by qualitative or quantitative values, like the link quality of a device or the accuracy of a sensor. An usdl:InteractionPoint is an actual step to be performed when accessing the service, like calling a CoAP based REST interface. usdl:ServiceOffering is an example for a connection to the business side. Service offerings may define a price, terms and conditions and SLAs.

C. Endpoint Description in Linked USDL

As shown in Figure 4 we "link" LUSDL to an endpoint vocabulary, thus adding endpoint information for calling the actual executable code on the smart object.

Each usdl:InteractionPoint is connected to one ore more endpoints. The endpoints are modeled using a Linked USDL extension called Linked USDL4IoT [15]. Each endpoint can have one or more operations. For each operation input/output parameters, as well as the actual implemented method to call is modeled.

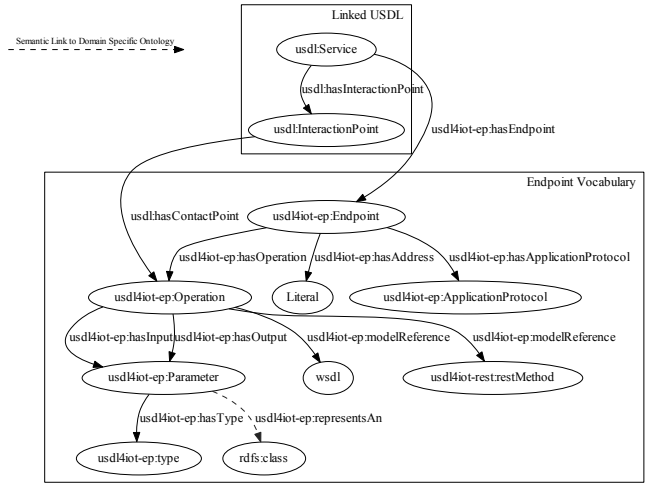

Fig. 4. LUSDL extended by an endpoint vocabulary [14]

It is notable, that endpoint descriptions are mainly used to express technical aspects, as well as the necessary surrounding semantics. A semantic link to domain specific ontologies is established through the yields and receives properties of an Interaction Point in LUSDL and the corresponding representsAn links on the input/output side.

We support REST and SOAP vocabularies, as well as operations that do not necessarily follow SOAP/REST and are modeled via a custom protocol. In such case the msm:Operation directly links to an URI endpoint, which can be either 6LoWPAN or UDP. These custom protocols are defined in ASN.1 [13]. A simple protocol for getting the temperate following the request/response pattern could look as follows:

```
TemperatureProtocol DEFINITIONS ::= BEGIN
  Request ::= SEQUENCE {
     sensorNo INTEGER
  }
  Response ::= SEQUENCE {
     temperature DOUBLE,
     timestamp INTEGER
  }
END
```

The code fragment defines a protocol (TemperatureProtocol) that accepts a sensor number as an integer and, in response, will deliver a temperature/timestamp pair. The actual encoding currently used is that of Java. Nonetheless, encodings like BER or XER [16] could easily be integrated as there exist several compilers for different programming languages.

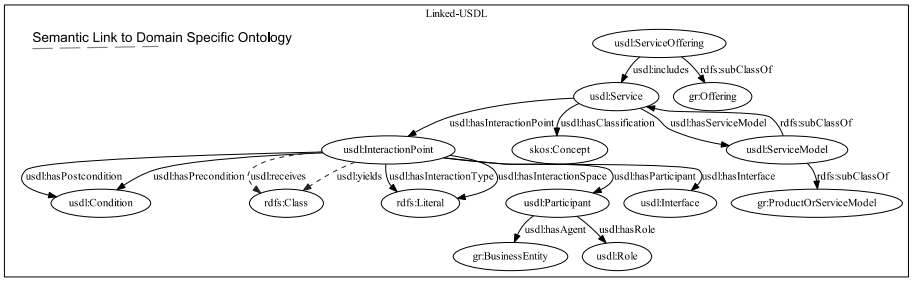

Fig. 2. Linked Unified Service Description Language [14] — only the relevant parts are shown.

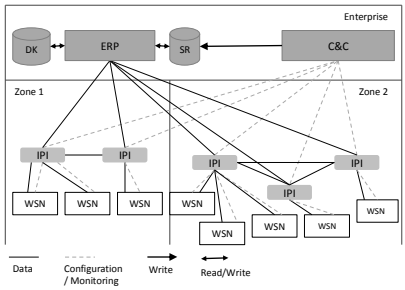

Fig. 3. Deployment

D. Communication Patterns

Traditionally, web service communication follows a request-response communication scheme. This is sufficient for some applications, but many IoT applications need more sophisticated communication patterns. In this work we support three types of communication patterns:

1) *Request/Response (R/S)*: The service consumer issues a request, which the service providers answers either synchronously or asynchronously.
2) *Publish/Subscribe (P/S)*: The publish/subscribe pattern allows a consumer to subscribe to particular events. As soon as this specific event occurs, the provider triggers a notification to the consumer. A subscriber based communication model is a necessary precondition for running parts of business processes on smart objects, as these often need to react on specific events.
3) *Time triggered*: While it can be argued that time-triggered is just a subclass of P/S in IoT there are many *sense and send* applications. This specific pattern is so common that it deserves to be a category of its own.

The R/S pattern is supported by LUSDL. For supporting events we add a small event vocabulary, called *usdl-event*. Based on existing studies of event-based systems [17] we identified the following subset of typical P/S operations which we support in usdl-event: (i) Register event subscriptions, (ii) Remove event subscriptions, (iii) Decouple resources from events and (iv) Specify means of delivery, e. g. callbacks.

We aim for compatibility with WS-Eventing, because such a mapping simplifies smart endpoint generation as outlined in Section V. Our schema also can be mapped to WS-Notification, and the semantic links to higher level ontologies matches the concepts introduced in WS-Topics very well.

V. Endpoint Generation

Enterprise systems typically have one or more service repositories. A service consumer queries the repository for the service to access and retrieves service descriptions and endpoints. Based on existing semantic service endpoint descriptions equivalent (alternative) endpoints can be created. This allows an application to choose from a set of endpoints, even when direct communication between the enterprise IT system and the smart object is not possible. As these descriptions live in the realm of the enterprise, the size of the description does not play a crucial role.

We propose the use of a graph transformation algorithm called *smart endpoint generation* (SEG). It takes an endpoint description with a known interface (e. g. CoAP) as input and transfers it into an equivalent interface using a different protocol or encoding. Another advantage of our SEG is the possibility to utilize given ontologies to guide the process of building the interface.

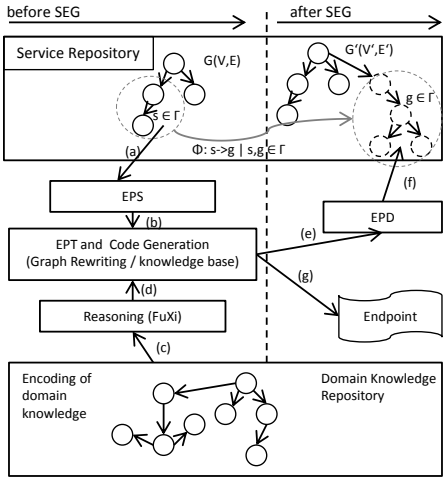

Fig. 5. Smart Endpoint Generation

In the following we take a closer look on how the SEG enables interoperability: When implementing our research prototype we did not build up transformations and knowledge bases for all possible combinations, but concentrated on those that seemed most interesting for us. On the smart object side we support REST Style CoAP with RDF and XML output representing the RDF-based semantic web stream of research. We also support REST HTTP XML as common in the web of things domain. As for specialized protocols we support ASN.1 specified protocols following a Request/Response pattern over MAC layer, over UDP or CoAP. On the IPI we support CoAP REST RDF, CoAP REST XML, SOAP WSDL, HTTP REST XML, and HTTP REST JSON. These represent currently the most frequently used protocols in enterprise computing.

The actual process of generating new endpoints is shown in Figure 5. Based on an already existing service in the service repository (a), which is read by the endpoint source handler (EPS) and then fed (b) into End Point Translation (EPT) and Code generation. The process of creating a new service endpoint is driven by a knowledge base and graph rewriting rules. It is, for example, possible to transform a graph representing a REST-based CoAP interface into a REST-based HTTP interface, thus providing a Web of Things like interface. This step results in a new endpoint description (e), which is then stored in the service repository. Furthermore, code for the actual endpoint can be created (g) by traversing the tree and applying templates stored in the knowledge base.

In the following we describe the structure of the algorithm which performs the endpoint translation in more detail:

The SEG-Algorithm is traversing a service description graph G(V, E) and transforming it into a a new service description graph G'(V', E') with one additional endpoint $g \in \Gamma$ from a set of possible endpoint types Γ.

For easier readability we assume that each node $v \in V$ has a unique type $type(v)$, which refers to the type of node, for instance "msm:Operation". For each interaction point IP_i, first a new endpoint g is added as a new interface to the corresponding interaction point. The source interface i is now traversed in pre-order and generates the interface i' on the fly. We defined for each type of a collection of transformation rules $\Phi(v \in V,\ s \in \Gamma,\ g \in \Gamma)$, which takes a node v, its original source endpoint type s and a requested destination endpoint type g. The actual transformation rule depends on type(v) and can be as easy as rewriting the URI for the newly created actual endpoint on the IPI, thus changing the CoAP protocol endpoint to an HTTP interface or creating an RDF output from an ASN.1 textual description. The latter one is supported by the yields attribute of the interaction point and the modelReference to guess an appropriate RDF subject/predicate/object triple.

The created endpoint g follows the paradigm of source endpoint s. In a transformation to WSDL, the resulting WSDL has the look and feel of a REST interface. Conversion from a WSDL to a REST interface is not possible.

Additionally, SEG can utilize given domain specific ontologies (c) to create endpoints tailored towards the business needs (d). This allows moving away from a pure device centric approach, in terms of sensor X senses property Y, to a more business entity based approach: The temperature of good G_i is Y_i, whose average temperature $AVG(temperature_i)$ can be measured by sensors $s_1..s_n$. In such a case, we know from the domain knowledge base (see Figure 1) that our service is related to a business entity G_i and that it currently is in site 1. We, furthermore, have the information that site 1 is a country which uses the Celsius scale for measuring these types of goods. We can now generate an endpoint for this specific good in the respective IPI and therefore allow the enterprise system to specifically ask for the temperature of good G_i. We leverage the Rete algorithm [18] to reason on the expected output value, which is encoded in the business entity. That way we know, that in case of good G we need to measure temperature. As both, the entity and the service, use the same ontology for temperature it is easily possible to reason which service to call.

VI. Implementation

We use a prototype implementation to show the feasibility of our approach. The enterprise level software is written in Java 7. The smart objects use the Moterunner (MR) platform [19] from IBM Research. The MR platform comes with a Java to custom byte code compiler (mrc), which can then be run on very constrained devices. Compared to earlier approaches (like, for example, Sun SPOTS) it has a very good tradeoff between using a VM and the energy consumption [19]. We created a custom CoAP-14 client/server implementation. On the Iris Motes we use the endpoint descriptions to generate code for specific services and, if necessary, add a small lightweight CoAP or HTTP library with minimal overhead. The smart endpoint generation tool (segen) is written in Python. Reasoning (forward chaining) is provided by FuXi - an open source library.

VII. Evaluation

We performed a quantitative evaluation of the proposed techniques to determine the additional costs of using RDF to get an idea how the different possibilities of implementing an endpoint influence the overall system. Therefore we describe the experimental setting, in which the evaluation was performed (VII-A); compare the sizes of different endpoint representations and compressions (VII-B); evaluate the communication between enterprise systems and motes. As we aim for easing interoperability between enterprise systems and sensor motes, it is essential to know the additional costs of higher level protocols (VII-C).

A. Experimental Setting

The experiments were performed in a living lab on IRIS Motes, with technical details as given in Table II.

We implemented an experimental retail situation, in which perishable goods are monitored and their price is adjusted according to the estimated customer demand and the remaining lifetime of the goods (dynamic pricing). The local enterprise application is running a triple store as a service repository and is connected to an instance of the integration platform. Motes can join and leave the network dynamically. The application is "location-aware", thus having a knowledge repository that contains information about the actual location of the goods as well as their main parameter (temperature in our case). The IPI is running on Raspberry Pi with 512MB RAM and an ARM1176JZF-S 700 Mhz CPU.

B. Endpoint Description

As we store either the complete or at least parts of a service description on the mote we first compared our on-mote RDF description with alternative descriptions in ASN.1 and WSDL. These are, of course, not on the same qualitative level, since the main purpose was to evaluate RDF compared to WSDL or ASN.1. The results can be seen in Figure 6: It is not surprising that ASN.1 is the most compact one, but, of course, does not come with any additional information for semantic M2M communication. In that case all the logic has to be in the application. The difference in resource consumption between ASN.1/WSDL and LUSDL is the price of semantic M2M. Due to the verbosity of XML, WSDL uncompressed is the largest one. We also tested several compression algorithms.

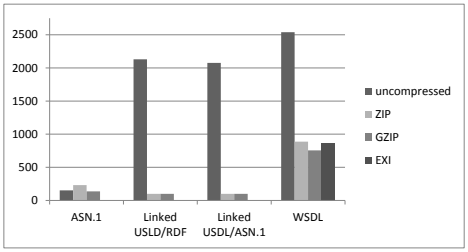

Fig. 6. Size of different endpoint description representations (in bytes)

It is notable that the general purpose algorithms performed better than algorithms tailored towards XML or RDF. Similar studies, but without looking into XML or RDF, also showed the advantages of general purpose compression schemes on wireless sensor notes [20]. As the service endpoints are rather small in size, the results with a full blown service description, resulting in a large XML is different. With increasing size of the XML file EXI outperforms ZIP and GZIP [21].

C. Enterprise to Mote Communication

When comparing the various means of an enterprise to access services via a generated endpoint we are mainly interested in measuring the performance and resource consumption, and compare that against direct access. Considering our use case introduced in Section VII-A we run the following evaluations:

1) Initial system setup time, this includes the time needed to recognize new motes, download the service description, generate an endpoint and add it to the repository.

ASN.1	CoAP	UDP	RUDP	MAC
Stack	256	210	241	101
Heap	2348	2290	2410	340
Flash	923	624	896	311
RDF	CoAP	UDP	RUDP	MAC
Stack	282	242	265	154
Heap	2401	2322	2470	387
Flash	935	657	929	332

TABLE I
RESOURCE CONSUMPTION (TOTAL, IN BYTES)

2) Accessing the service via CoAP over 6LoWPAN
3) Accessing the service via an alternative endpoint with temperature conversion

The initial setup time of motes connecting to the system consists of the time the 6LoWPAN implementation needs till the mote is recognized. This is solved by the protocol itself, which notifies the edge mote about any joining or leaving mote. Implementation details can be found in [22]. The initial setup time for our system ranged from less than 5 seconds, up to 30 seconds depending on the traffic in the sensor network and interference caused by wireless LAN and GSM devices. As service descriptions are larger and need more transmission time they are more prone to packet loss.

We evaluated the behaviour of the system when accessed from a gateway. First we accessed the service with CoAP over UDP and 6LoWPAN over one, two and five hops. We then did the same with the protocol specified in ASN.1 over UDP, and once more over a custom written UDP based end-to-end reliability protocol (RUDP) with sequence numbers and acknowledgements. CoAP was used in reliable mode, thus all requests were acknowledged. In all cases we measured the time including serialization and deserialization on the client side. The timeout values of the RUDP layer request/response were set to twice the CoAP timeout of 400ms, after which a packet was considered lost and the request failed. The rate of unsuccessful requests with more than two subsequent lost packages was less than 0.001%.

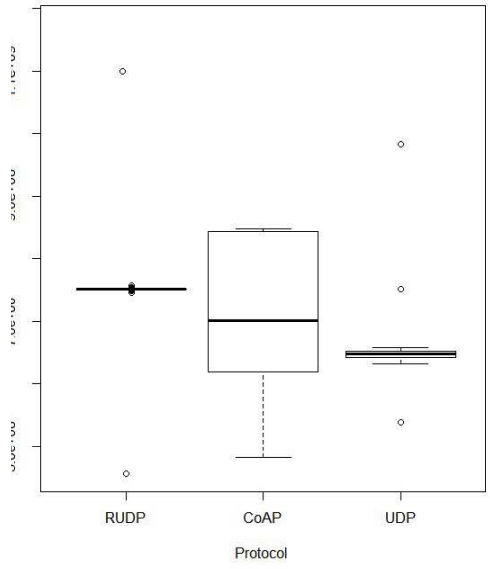

Fig. 7. Measurements Gateway to Mote (in ns)

The results for the two hop scenario are shown in Figure 7. All other scenarios showed basically the same characteristics. It is notable that CoAP has a much higher variation in response time than UDP or RUDP, because of the higher protocol complexity by an additional layer of information to be processed. Generally speaking, UDP was slightly more performant than CoAP, while RUDP was comparable to the median CoAP response times.

The resource usage of the respective code is shown in Table I. All values are maximum numbers after 2000 requests gathered with the MR profiler. As expected, with higher protocol complexity the resource usage increases. Nonetheless, CoAP does not need considerably more space than a pure UDP solution and comes with end-to-end reliability. Our RUDP did not use much less resources than the CoAP implementation.

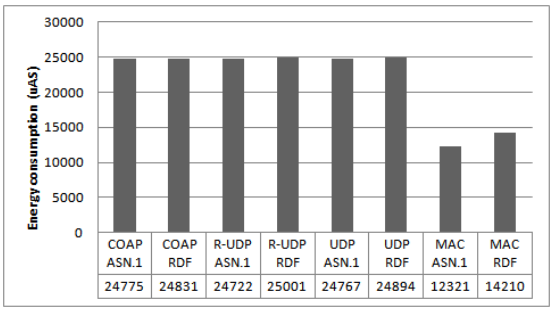

Fig. 8. Energy consumption (in μAS, 10min)

Energy consumption is shown in Figure 8. All measurements are given in μAS. Our results are comparable to previous work done with equally complex software [23]. As expected, the 6LoWPAN layer contributes most to energy consumption, as it can be seen in the difference between UDP/CoAP and the MAC level communication. The additional power needed for implementing the CoAP layer is less than 1%. A reliable UDP implementation other than CoAP needs nearly the same energy.

Once the service description was downloaded, we measured the Service Access Time (SAT), i. e., the time from issuing a service request by the service consumer until the response has been decoded and the data is ready to be used. As benchmark we used a CoAP-based REST style protocol on the mote as well as just sending the temperature and the time stamp. We then measured Service Access Times times for (i) direct access on the same machine via IPv6, (ii) remote access from a different machine over the local network over IPv6, (iii) Remote Access via generated SOAP and (iv) HTTP REST interfaces. The results are shown in Figure 9.

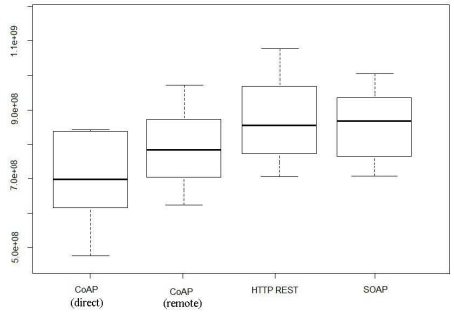

Fig. 9. Service access times (in ns, per protocol)

Direct access is, of course, the fastest method, directly followed by access over the local network (no other traffic). More

CPU	ATmega1281@8 Mhz, 3V	RF	IEEE 802.15.4
Flash	512kB (serial), 128kB (program)	RAM	8k bytes
Current	8mA(act), 8μA(slp)		

TABLE II
TECHNICAL DETAILS OF AN IRIS MOTE

interesting is to quantify the additional costs of accessing the services on the mote: REST access over http was slightly faster than SOAP due to reduced protocol stack complexity. The extra work of converting the temperature did not contribute significantly to the Service Access Time.

VIII. CONCLUSIONS

We have shown that semantic endpoint descriptions in conjunction with smart endpoint generation can be used to efficiently integrate IoT-based smart objects into enterprise platforms. Compared to existing solutions we integrated not only ways of establishing technical interoperability, like pure RDF or WSDL does, but support also legacy protocols or protocols with special requirements as often found in sensor network environments. For example, in settings where energy consumption is a major issue, the semantic service descriptions with ASN.1 combined with smart endpoints can be used to ensure interoperability.

One of the main challenges of innovation in the enterprise industry today is to cope with the myriad of already existing code. Vendors, therefore, aim towards innovation that has a clear integration path into existing systems (timeless software). As most enterprise SOA systems already use service repositories as an integrated part of their SOA environment, the integration of semantic service descriptions would not change the paradigm of how software is written today and thus can be added to enterprise software in an incremental way without the need of disruptional changes. Our platform eases the enterprise integration and interoperability by providing high level application designers with the same means of accessing a smart object as they access another enterprise system.

When looking into establishing interoperability none of the options has any severe drawback that would lead to a definitive exclusion from further consideration. Legacy enterprise systems could be integrated by providing a REST or SOAP-style interface without having to write any further custom code and nearly without manual work, because endpoints for these systems can be created automatically. In the SOAP case the access time would, for example, increase by around 4.2ms, compared to direct access via CoAP. Caching of sensor data would even decrease that further. CoAP proved to consume not much more energy than a pure UDP solution written for just one purpose. In special cases, where energy efficiency is the primary objective, dedicated protocols including lower levels have to be considered.

Emerging web of data applications can be supported by either creating RDF triples on the alternative endpoint or by sending them directly. On the alternative endpoint there is then the possibility to add further information like the entity to which the sensor is attached or the geolocation to provide higher level services to an enterprise system [24].

We have shown the feasibility of our approach in a real world setting. Our evaluation has shown that endpoint descriptions in RDF do not need considerable more memory than SOAP or XML REST based services and evaluated the price in terms of energy for application level protocols and latency for introducing another layer of indirection. Moreover, we demonstrated the flexibility that comes with semantic endpoints and smart endpoint generation. These allow to describe arbitrary services, including input and output parameters and are easily transformable into other endpoint formats. Furthermore, they can be used to decouple sensor data from actual sensing devices and provide higher level services to enterprise systems, allowing not only a device centric view on sensor data, but also a business entity based view, which than can be integrated into enterprise systems.

ACKNOWLEDGMENT

The research on this topic received funding from the EC under grant 257521 (IOT-A) and grant 285248 (FI-WARE). We would like to thank the Moterunner Team at IBM Research, especially Marcus Oestreicher as well as our students Theano Mintsi and Michael Gede for their valuable support.

REFERENCES

[1] Z. Shelby and C. Bormann, *6LoWPAN: the wireless embedded internet*. Wiley, 2011, vol. 43.
[2] Z. Shelby, K. Hartke *et al.*, "Constrained Application Protocol, Internet-Draft," 2011.
[3] S. Haller, S. Karnouskos *et al.*, "The Internet of Things in an Enterprise Context," in *Future Internet FIS 2008*, ser. Lecture Notes in Computer Science. Springer, 2009, vol. 5468.
[4] N. Glombitza, D. Pfisterer *et al.*, "Integrating wireless sensor networks into web service-based business processes," in *Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*. New York: ACM, 2009.
[5] D. Guinard, V. Trifa *et al.*, "A resource oriented architecture for the web of things," *Proc. of IoT*, 2010.
[6] C. Pedrinaci and J. Domingue, "Toward the Next Wave of Services: Linked Services for the Web of Data," *Journal of Universal Computer Science*, vol. 16, no. 3, 2010.
[7] T. Leidig and C. Pedrinaci, "Linked USDL - Development version," 2013. [Online]. Available: https://github.com/linked-usdl/usdl-core
[8] P. Barnaghi, M. Presser *et al.*, "Publishing linked sensor data," in *CEUR Workshop Proceedings: Proceedings of the 3rd International Workshop on Semantic Sensor Networks (SSN)*, 2010.
[9] J. Kopecky, T. Vitvar *et al.*, "Sawsdl: Semantic annotations for wsdl and xml schema," *Internet Computing, IEEE*, vol. 11, no. 6, 2007.
[10] J. Kopecky, K. Gomadam *et al.*, "hrests: An html microformat for describing restful web services," in *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08*, vol. 1. IEEE, 2008.
[11] D. Martin, M. Paolucci *et al.*, "Bringing semantics to web services: The owl-s approach," in *Semantic Web Services and Web Process Composition*. Springer, 2005.
[12] I. K. Samaras, J. V. Gialelis *et al.*, "Integrating wireless sensor networks into enterprise information systems by using web services," in *Sensor Technologies and Applications. Conference on*. IEEE, 2009.
[13] ITU, "Abstract Syntax Notation One: Specification of Basic Notation," ITU-T Rec. X.680, July 2002.
[14] T. Leidig and C. Pedrinaci, "Linked USDL," 2012. [Online]. Available: http://www.linked-usdl.org
[15] Thoma, Matthias and Braun, Torsten and Sperner, Klaus and Magerkurth, Carsten, "Linked USDL4IoT," Tech. Rep., 2014. [Online]. Available: http://www.iot4bpm.de
[16] ITU, "Information Technology — ASN.1 Encoding Rules," ITU-T Recommendation X.690, July 2002.
[17] Y. Huang and D. Gannon, "A comparative study of web services-based event notification specifications," in *Parallel Processing Workshops, ICPP Workshops. International Conference on*. IEEE, 2006.
[18] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial intelligence*, vol. 19, no. 1, 1982.
[19] A. Caracas, C. Lombriser *et al.*, "Energy-efficiency through micro-managing communication and optimizing sleep," in *Sensor, Mesh and Ad Hoc Communications and Networks, Conference on*. IEEE, 2011.
[20] K. Dolfus and T. Braun, "An evaluation of compression schemes for wireless networks," in *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010*. IEEE, 2010.
[21] D. Peintner, H. Kosch *et al.*, "Efficient XML Interchange for rich internet applications," in *Multimedia and Expo, ICME. IEEE International Conference on*. IEEE, 2009.
[22] IBM Research, "Moterunner SDK Beta 11 Documentation — 6LoWPAN implementation details," 2013.
[23] A. Caracas and A. Bernauer, "Compiling business process models for sensor networks," in *Distributed Computing in Sensor Systems and Workshops, International Conference on*. IEEE, 2011.
[24] M. Thoma, S. Meyer *et al.*, "On IoT-services: Survey, Classification and Enterprise Integration," in *Internet of Things (iThings), 2012 IEEE Conference on*. IEEE, 2012.